Überblick über die Roblet®-Technik

Hagen Stanek

gefördert durch
genRob

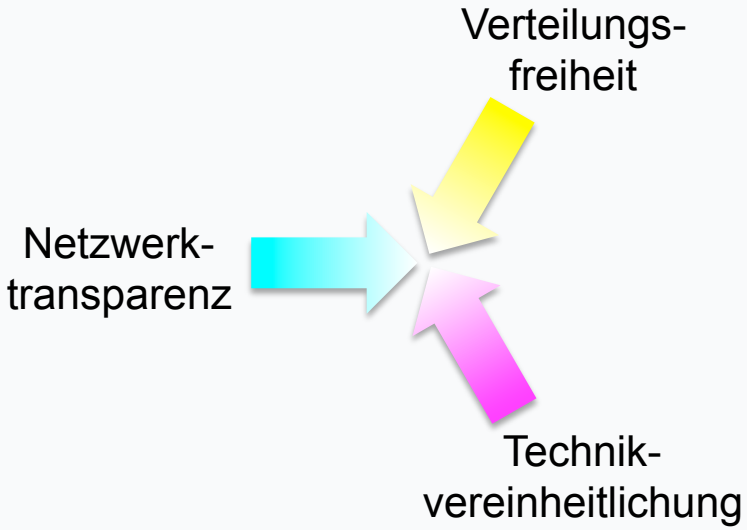
Agenda

- Teil 1: Motivation, Einführung
- Teil 2: Beispiel-Vorführung
- Teil 3: Anwendungen, Zusammenfassung

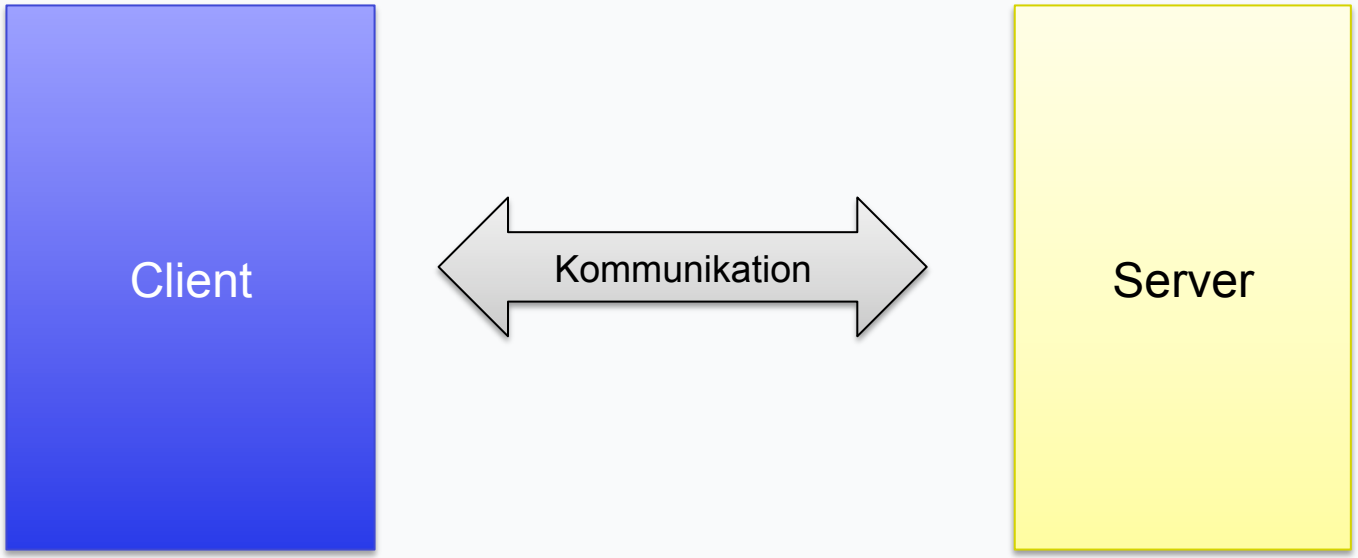
Teil 1

Motivation, Einführung

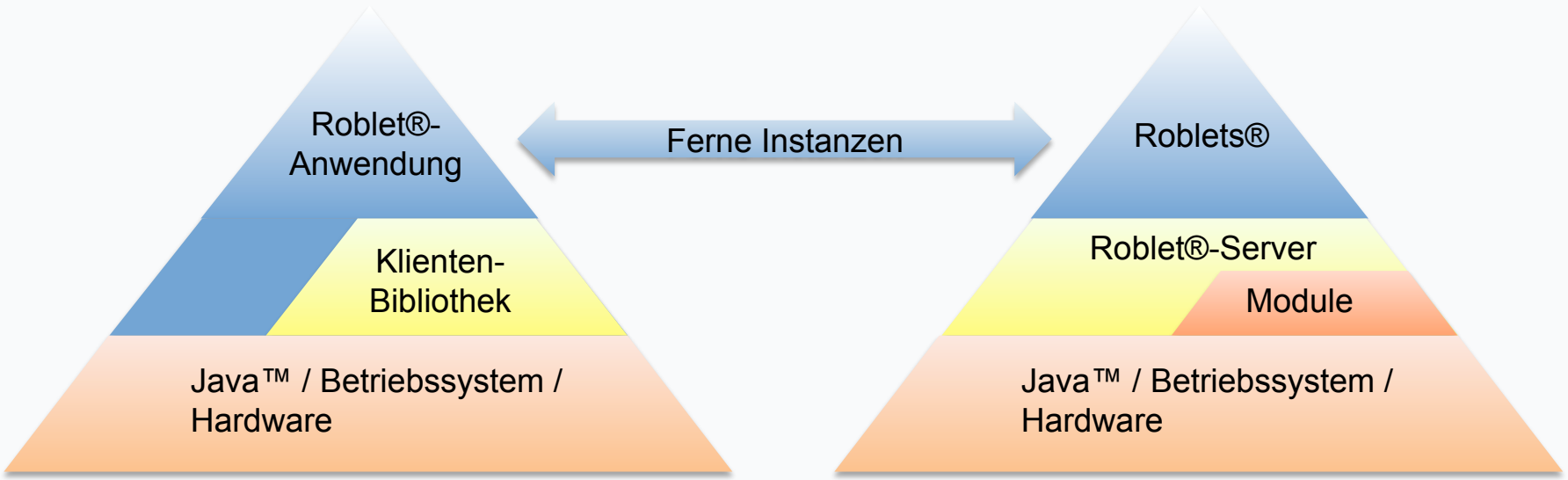
Teil 1 – Motivation – Effizienzdreieck



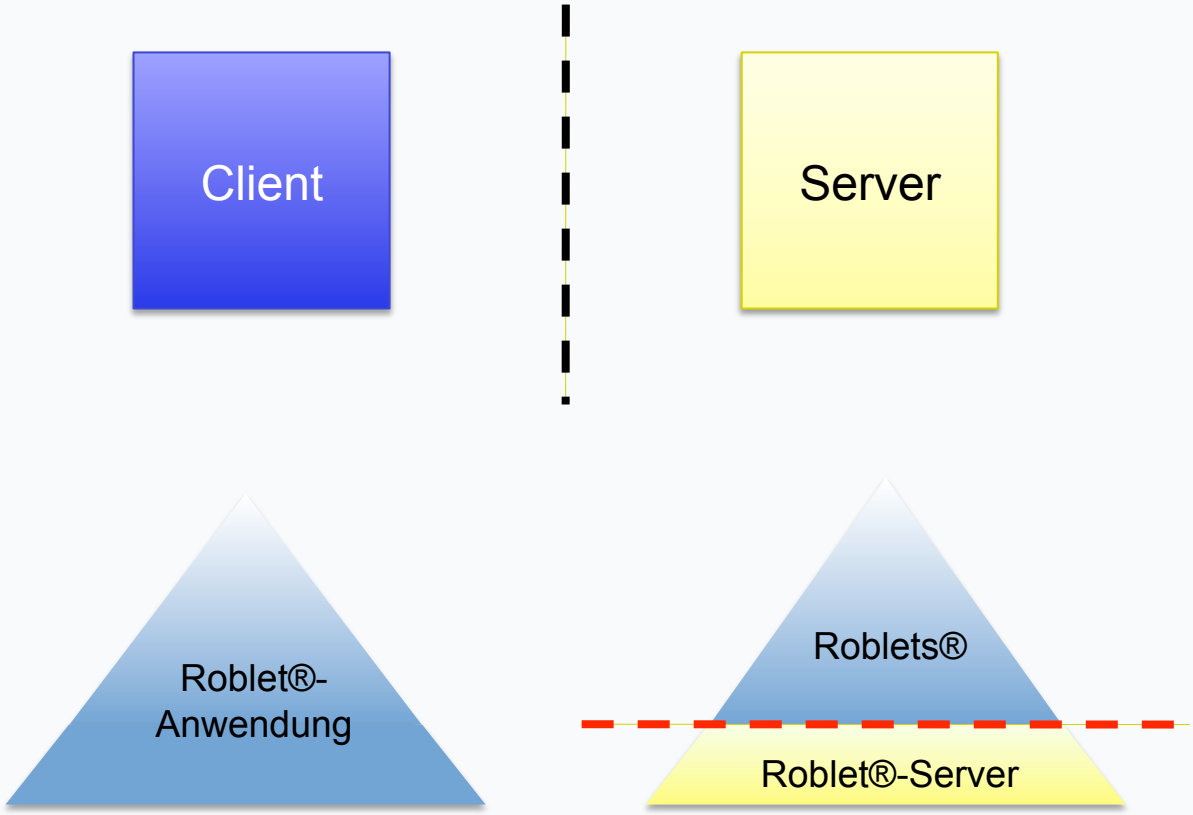
Teil 1 – Motivation – Client/Server



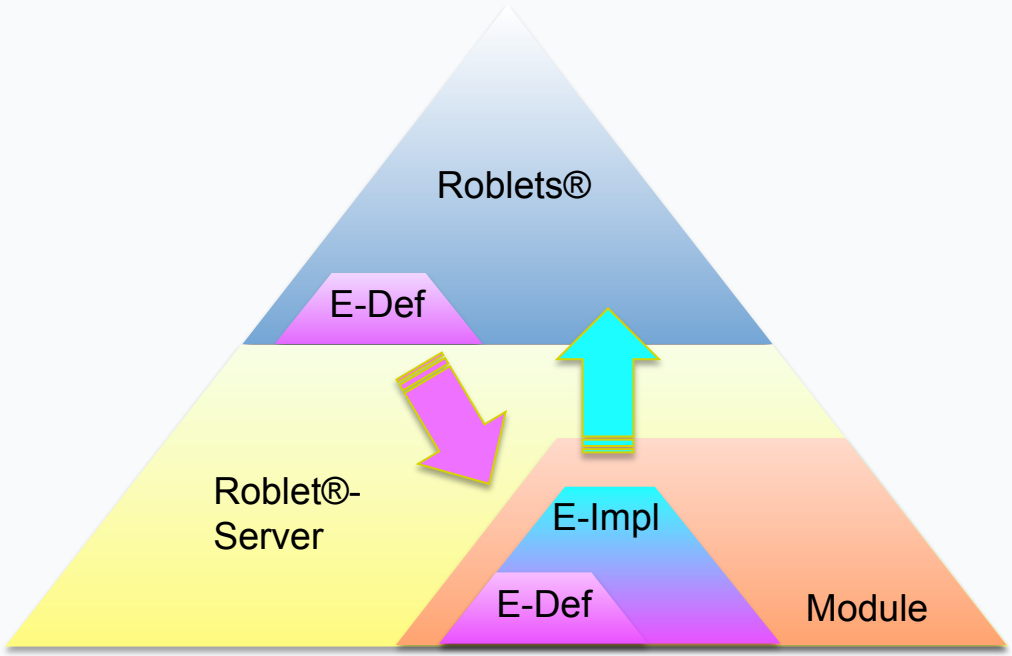
Teil 1 – Einführung – Architektur



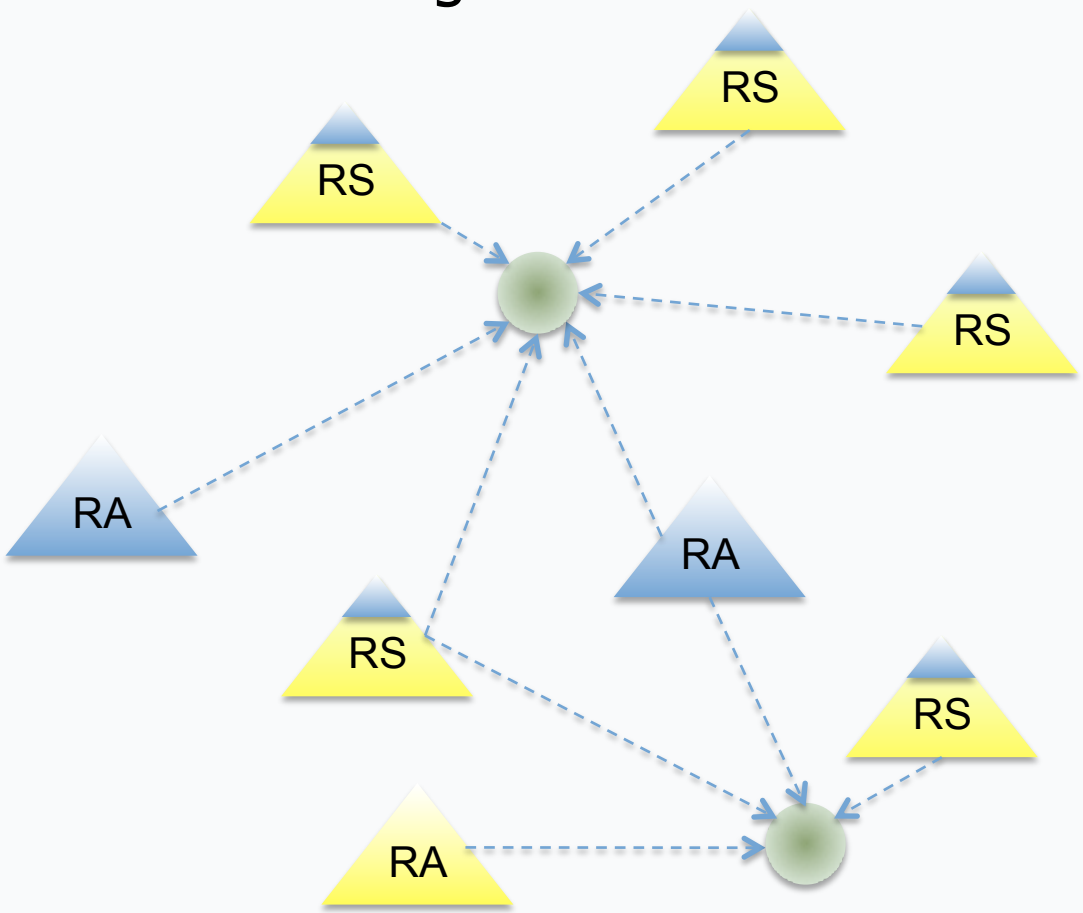
Teil 1 – Einführung – Schnittstellen-Vergleich



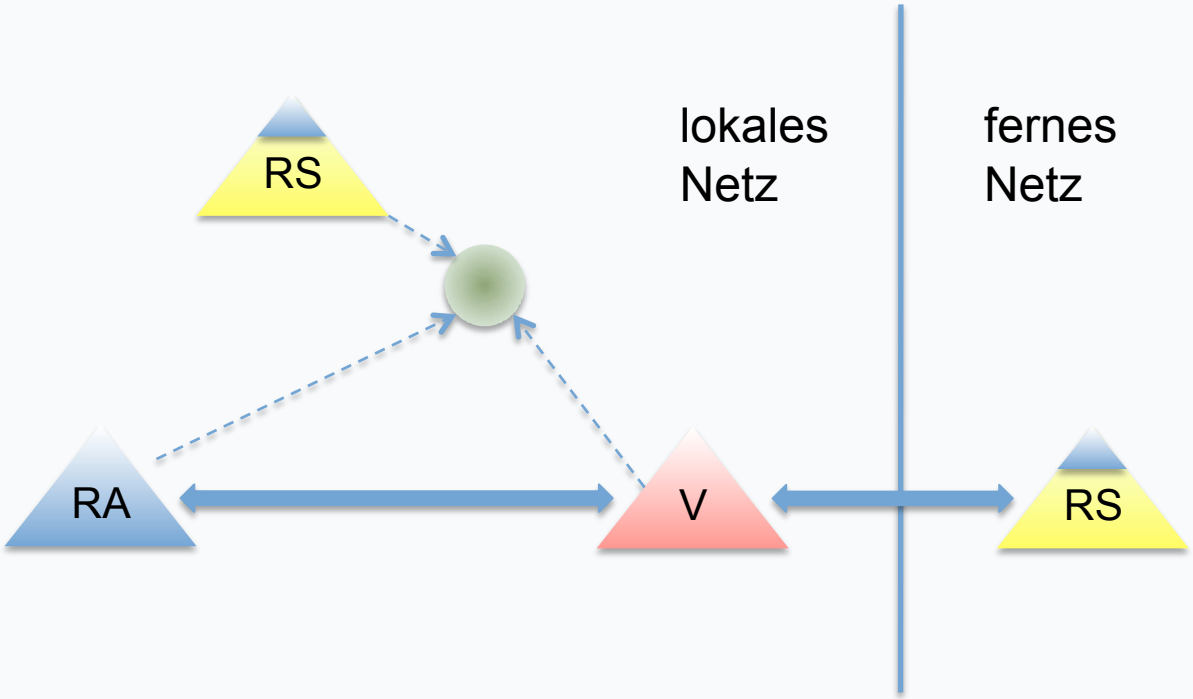
Teil 1 – Einführung – Einheiten



Teil 1 – Einführung – Verzeichnisse



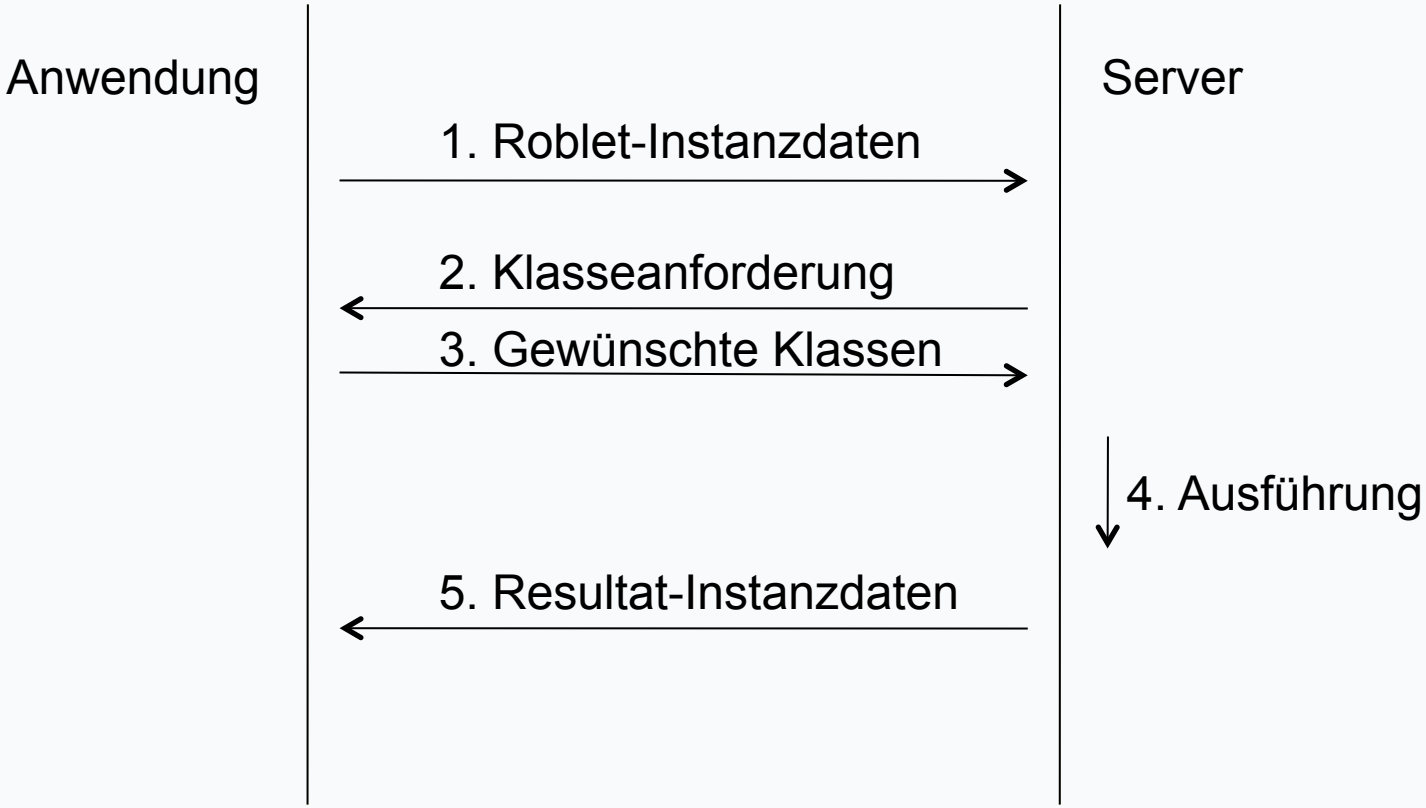
Teil 1 – Einführung – Vertreter



Teil 1 – Motivation – Hallo Welt!

```
public class  HelloApp
{
    public static void  main (String[] args)   throws Exception
    {
        System.out.println ("Hallo - lokal");
        new Client (). getServer ("roblet.org:2001").
                        getSlot (). run (new HelloRoblet ());
    }
}

class  HelloRoblet   implements Roblet, Serializable
{
    public Object  execute (Robot robot)
    {
        System.out.println ("Hallo - fern");
        return null;
    }
}
```

Teil 1 – Einführung – Zeitleiste



Teil 1 – Einführung – Nutzen

- Effizient bei der Entwicklung
  - Arbeiten in Java™ in allen Teilen (**Technikvereinheitlichung**) inkl. Ausnahmentransport (Fehlerübermittlung)
  - plus **Netzwerktransparenz**
    - durch automatischem Klassentransport
    - automatischem Verbindungswiederaufbau mit optionalem Abbruch
    - Unabhängigkeit von NAT oder Firewall
  - Einfaches verschieben von Funktionalität (**Verteilungsfreiheit**)
- Effizient im Netz durch eine einzige TCP-Sitzung pro Anwendung und Server bei beliebig vielen Fächern, Roblets® und fernen Instanzen
- Netzwerk-Protokoll in Klienten-Bibliothek bzw. im Server in Java™ definiert – muß von niemandem implementiert werden

Teil 1 – Einführung – Vergleich zu RMI

- Viel einfacher in der Handhabung gegenüber RMI
  - benötigt separate Server für Klassenladen
  - mind. 2 bei Roblet-ähnlicher Konfiguration
  - Standard ist HTTP, also Mehraufwand bei Sicherheitsbedarf
  - Jede Klasse wird einzeln per TCP-Sitzung geholt oder gar ganze JAR's
  - Ohne deutlichen Mehraufwand Probleme mit NAT, Firewall
- RMI bietet keine Laufzeitumgebung
  - Roblet nicht von Ressourcen getrennt
  - Es kann keinen unkooperativen Code beenden
- Rückrufe sind separate TCP-Sitzung(en)
- Kein RMIC nötig

Teil 1 – Einführung – Vergleich zu Distributed OSGi

- OSGi prinzipiell keine Code-Verteilung, sondern Dienstverwaltung in einer JVM
- OSGi muß Problem der Distribution durch separate Mechanismen lösen
- OSGi erfordert selbst für kleine Anwendungen Konfigurationsdateien und eine vergleichsweise aufwendige Struktur
- D-OSGi verteilt Informationen über angebotene Dienste
- D-OSGi ermöglicht Aufruf entfernter Dienste nach Art eines RPC
- Kein Code-Transport bei D-OSGi
- Keine Instanz-Daten

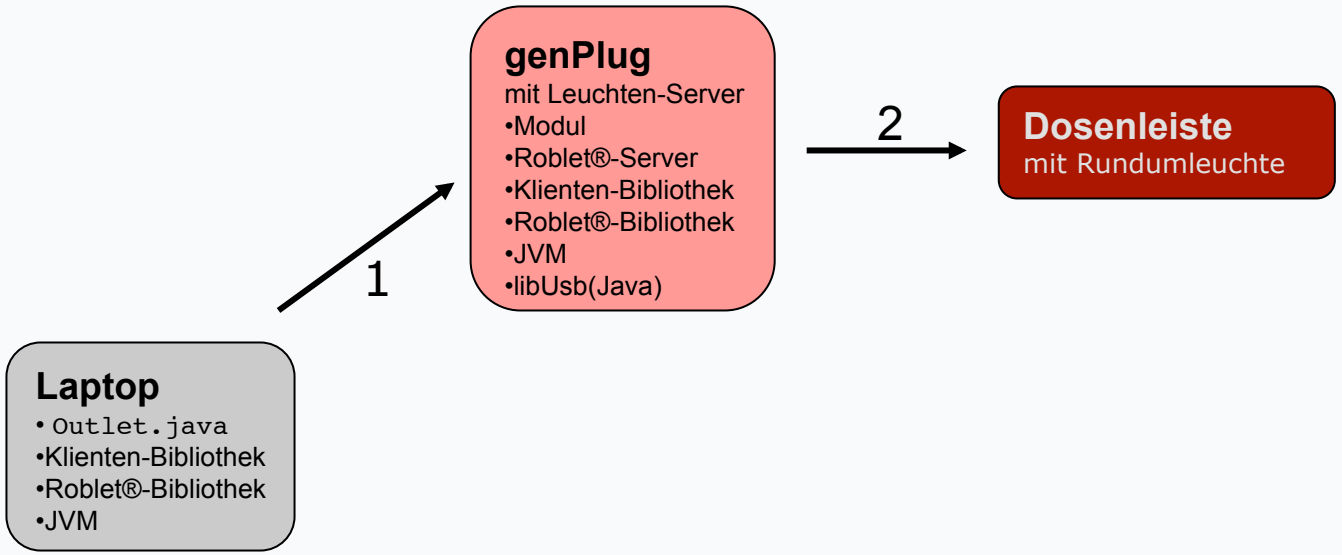
Teil 1 – Einführung – Vergleich zu Web-Java

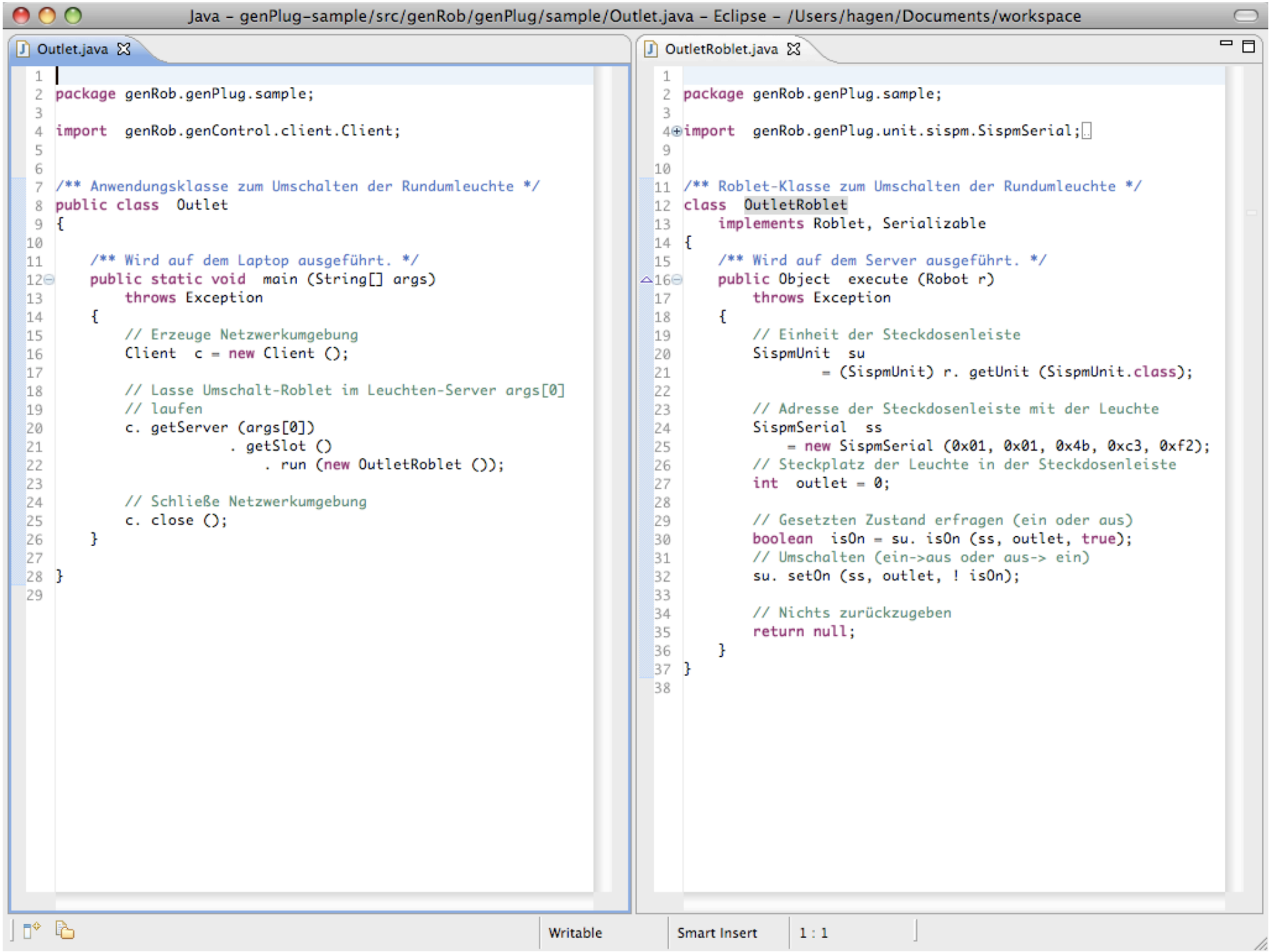
- Gemeint: Applets und Java Web Start
- Code-Pull (statt Code-Push bei Roblet®-Technik)
- Ganze Java-Archive und –Ressourcen
- Keine Instanzdaten (eingeschränkt bei Applets)
- TCP-Sitzung pro Datei
- Einrichtungsaufwand in Webserver(n)

Teil 2

Beispiel-Vorführung

Teil 2 – Rundumleuchte schalten (`Outlet.java`)

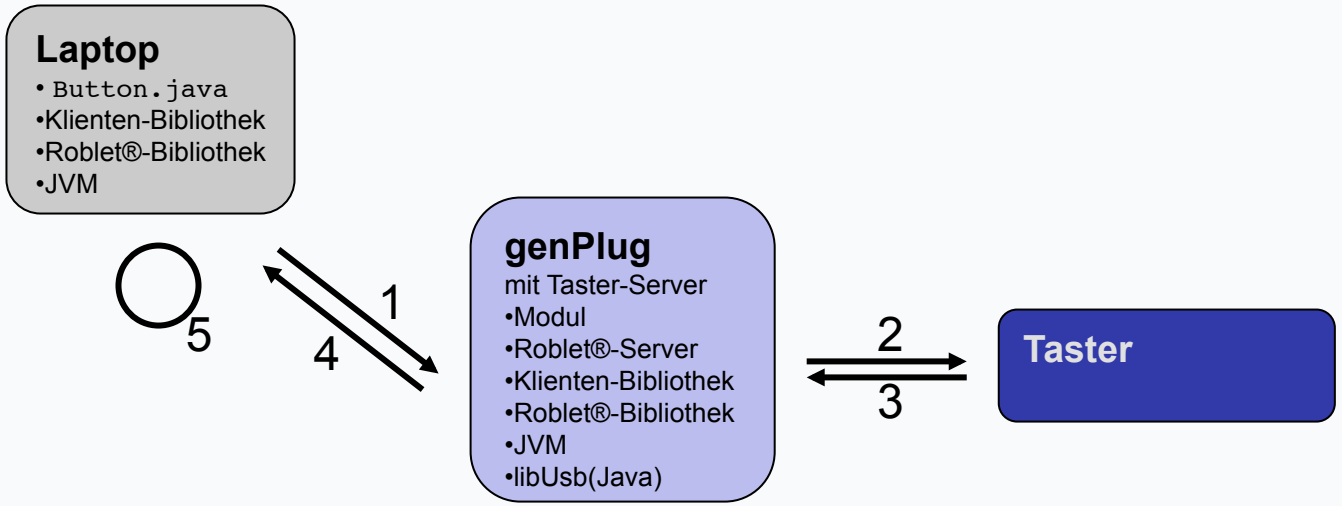




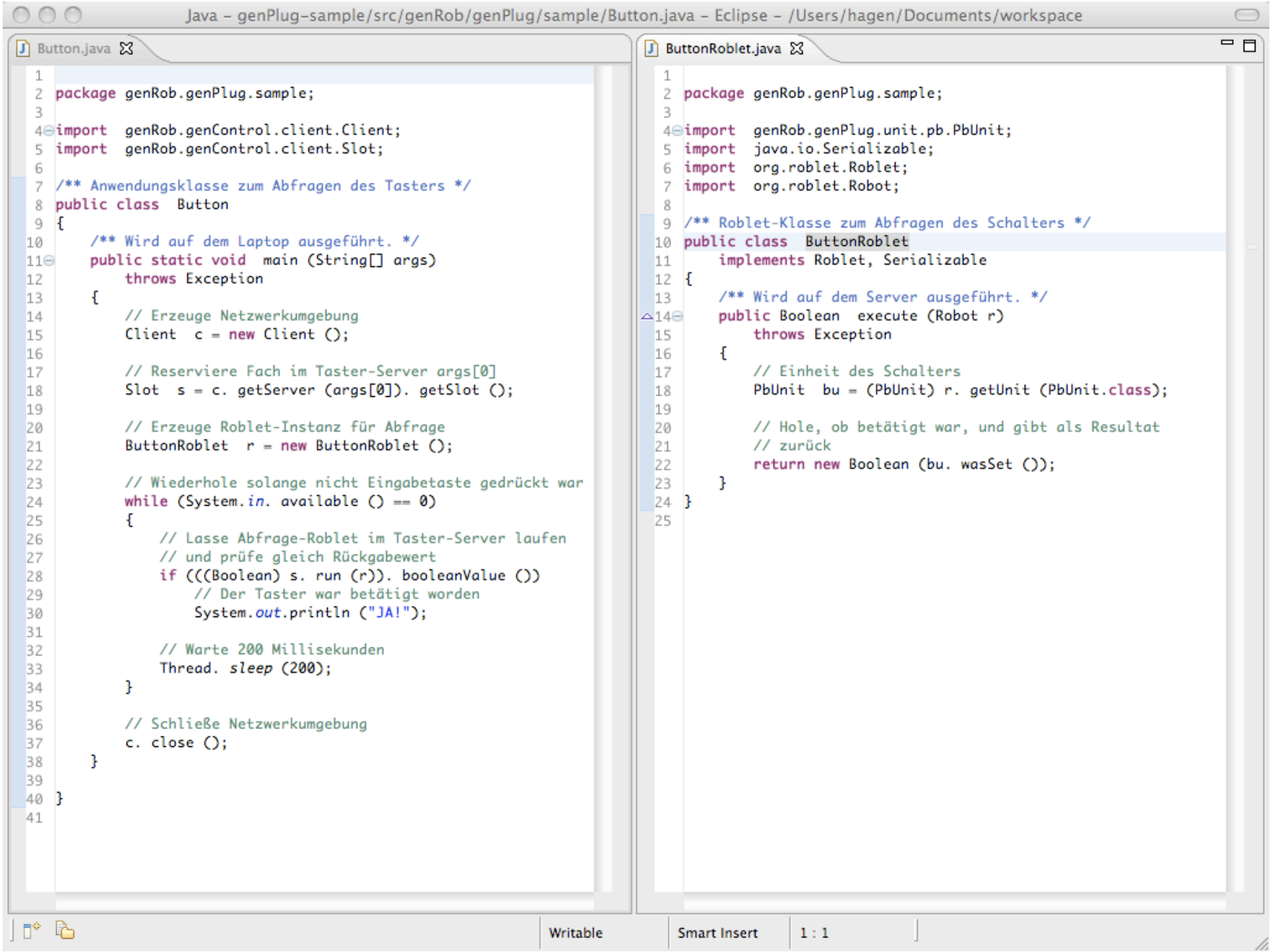

```java
package genRob.genPlug.sample;

import  genRob.genControl.client.Client;


/** Anwendungsklasse zum Umschalten der Rundumleuchte */
public class  Outlet
{

    /** Wird auf dem Laptop ausgeführt. */
    public static void  main (String[] args)
        throws Exception
    {
        // Erzeuge Netzwerkumgebung
        Client  c = new Client ();

        // Lasse Umschalt-Roblet im Leuchten-Server args[0]
        // laufen
        c. getServer (args[0])
                    . getSlot ()
                        . run (new OutletRoblet ());

        // Schließe Netzwerkumgebung
        c. close ();
    }

}
```

```java
package genRob.genPlug.sample;

import  genRob.genPlug.unit.sispm.SispmSerial;


/** Roblet-Klasse zum Umschalten der Rundumleuchte */
class  OutletRoblet
    implements Roblet, Serializable
{
    /** Wird auf dem Server ausgeführt. */
    public Object  execute (Robot r)
        throws Exception
    {
        // Einheit der Steckdosenleiste
        SispmUnit  su
                = (SispmUnit) r. getUnit (SispmUnit.class);

        // Adresse der Steckdosenleiste mit der Leuchte
        SispmSerial  ss
            = new SispmSerial (0x01, 0x01, 0x4b, 0xc3, 0xf2);
        // Steckplatz der Leuchte in der Steckdosenleiste
        int  outlet = 0;

        // Gesetzten Zustand erfragen (ein oder aus)
        boolean  isOn = su. isOn (ss, outlet, true);
        // Umschalten (ein->aus oder aus-> ein)
        su. setOn (ss, outlet, ! isOn);

        // Nichts zurückzugeben
        return null;
    }
}
```

Teil 2 – Notschalter abfragen (`Button.java`)



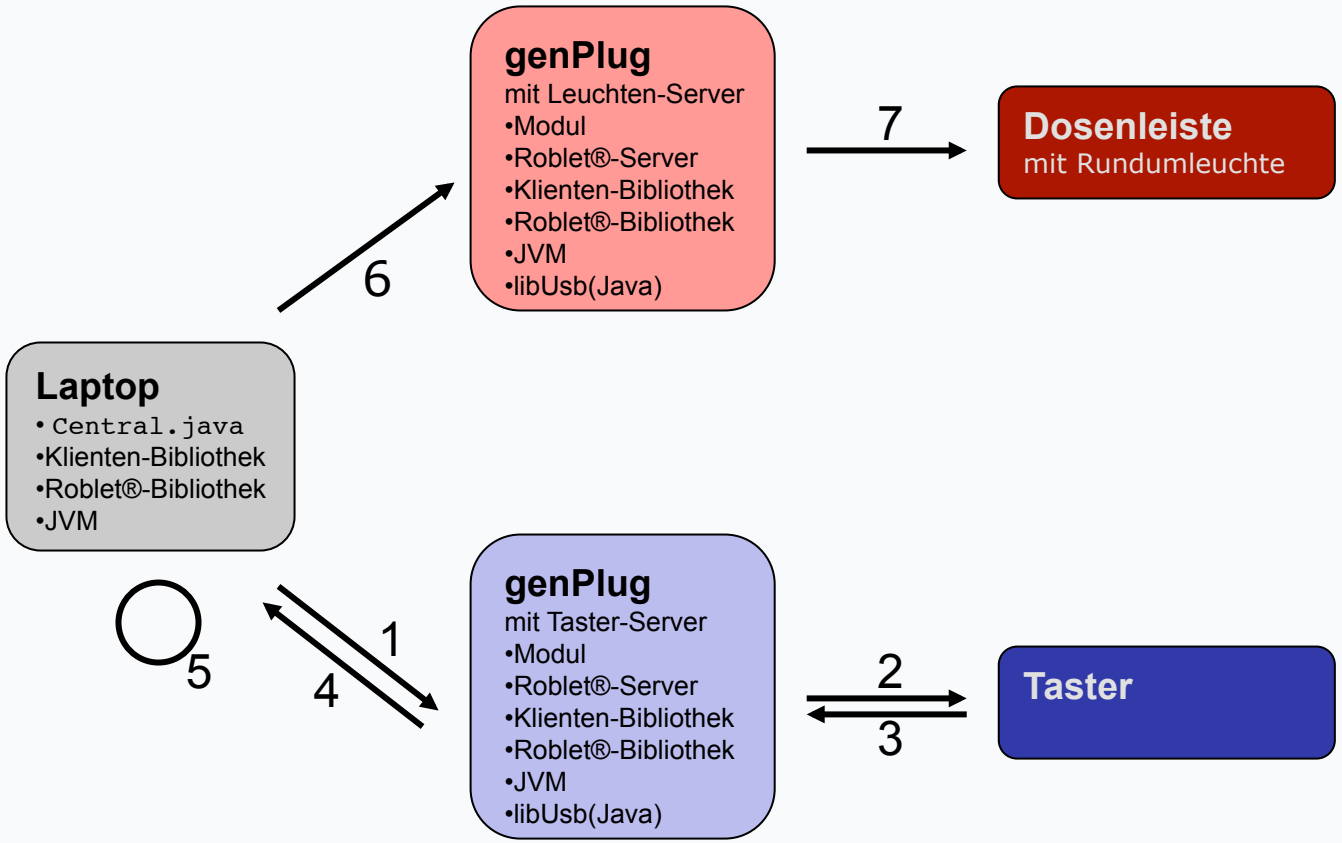
http://genRob.com/genPlug

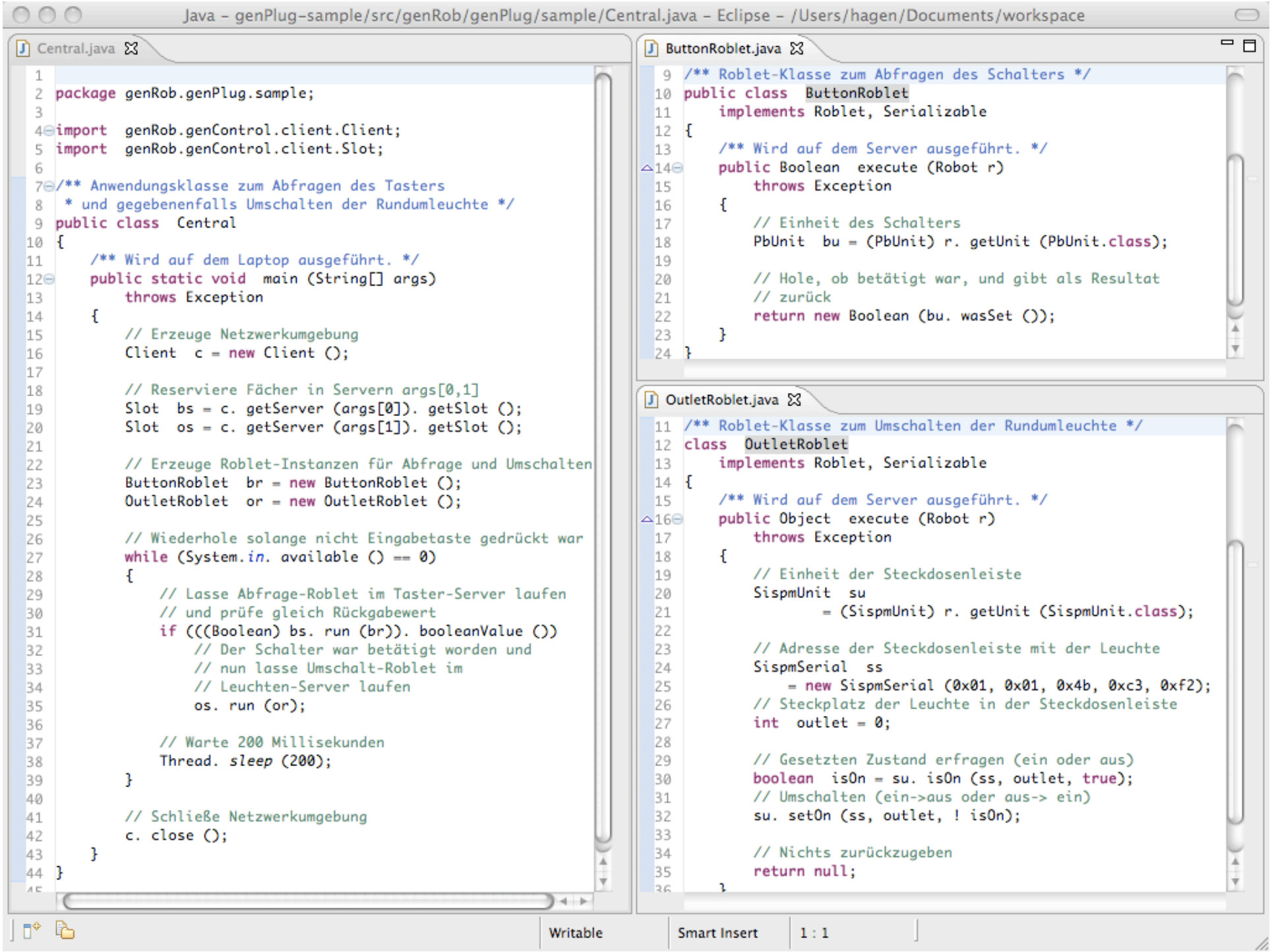



```java
package genRob.genPlug.sample;

import  genRob.genControl.client.Client;
import  genRob.genControl.client.Slot;

/** Anwendungsklasse zum Abfragen des Tasters */
public class  Button
{
    /** Wird auf dem Laptop ausgeführt. */
    public static void  main (String[] args)
        throws Exception
    {
        // Erzeuge Netzwerkumgebung
        Client  c = new Client ();

        // Reserviere Fach im Taster-Server args[0]
        Slot  s = c. getServer (args[0]). getSlot ();

        // Erzeuge Roblet-Instanz für Abfrage
        ButtonRoblet  r = new ButtonRoblet ();

        // Wiederhole solange nicht Eingabetaste gedrückt war
        while (System.in. available () == 0)
        {
            // Lasse Abfrage-Roblet im Taster-Server laufen
            // und prüfe gleich Rückgabewert
            if (((Boolean) s. run (r)). booleanValue ())
                // Der Taster war betätigt worden
                System.out.println ("JA!");

            // Warte 200 Millisekunden
            Thread. sleep (200);
        }

        // Schließe Netzwerkumgebung
        c. close ();
    }

}
```

```java
package genRob.genPlug.sample;

import  genRob.genPlug.unit.pb.PbUnit;
import  java.io.Serializable;
import  org.roblet.Roblet;
import  org.roblet.Robot;

/** Roblet-Klasse zum Abfragen des Schalters */
public class  ButtonRoblet
    implements Roblet, Serializable
{
    /** Wird auf dem Server ausgeführt. */
    public Boolean  execute (Robot r)
        throws Exception
    {
        // Einheit des Schalters
        PbUnit  bu = (PbUnit) r. getUnit (PbUnit.class);

        // Hole, ob betätigt war, und gibt als Resultat
        // zurück
        return new Boolean (bu. wasSet ());
    }
}
```

Teil 2 – Zentrale Steuerung (`Central.java`)



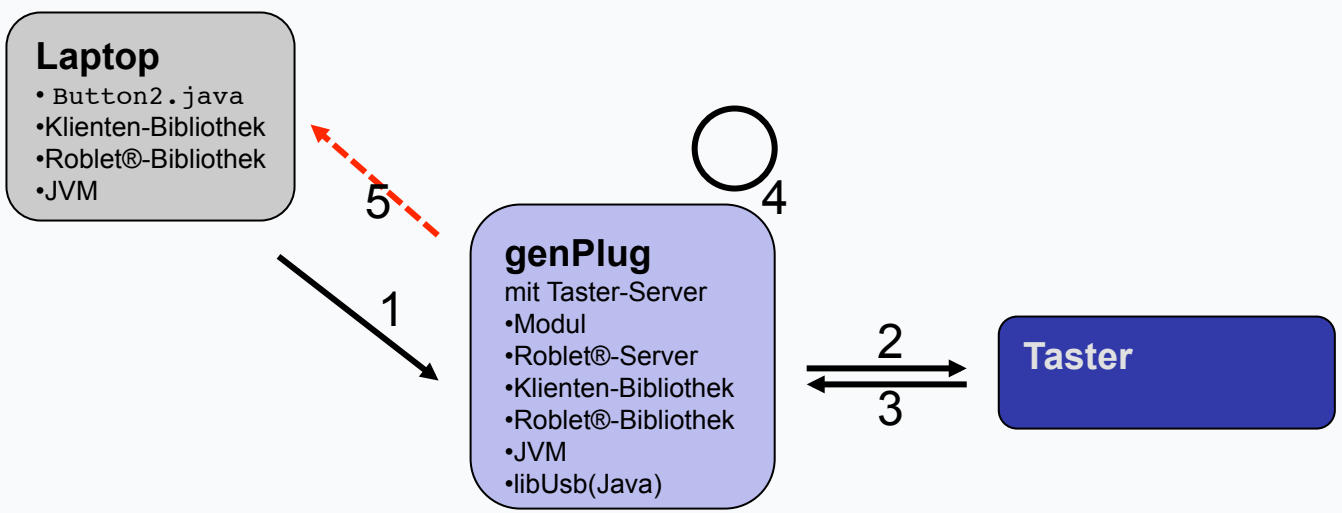


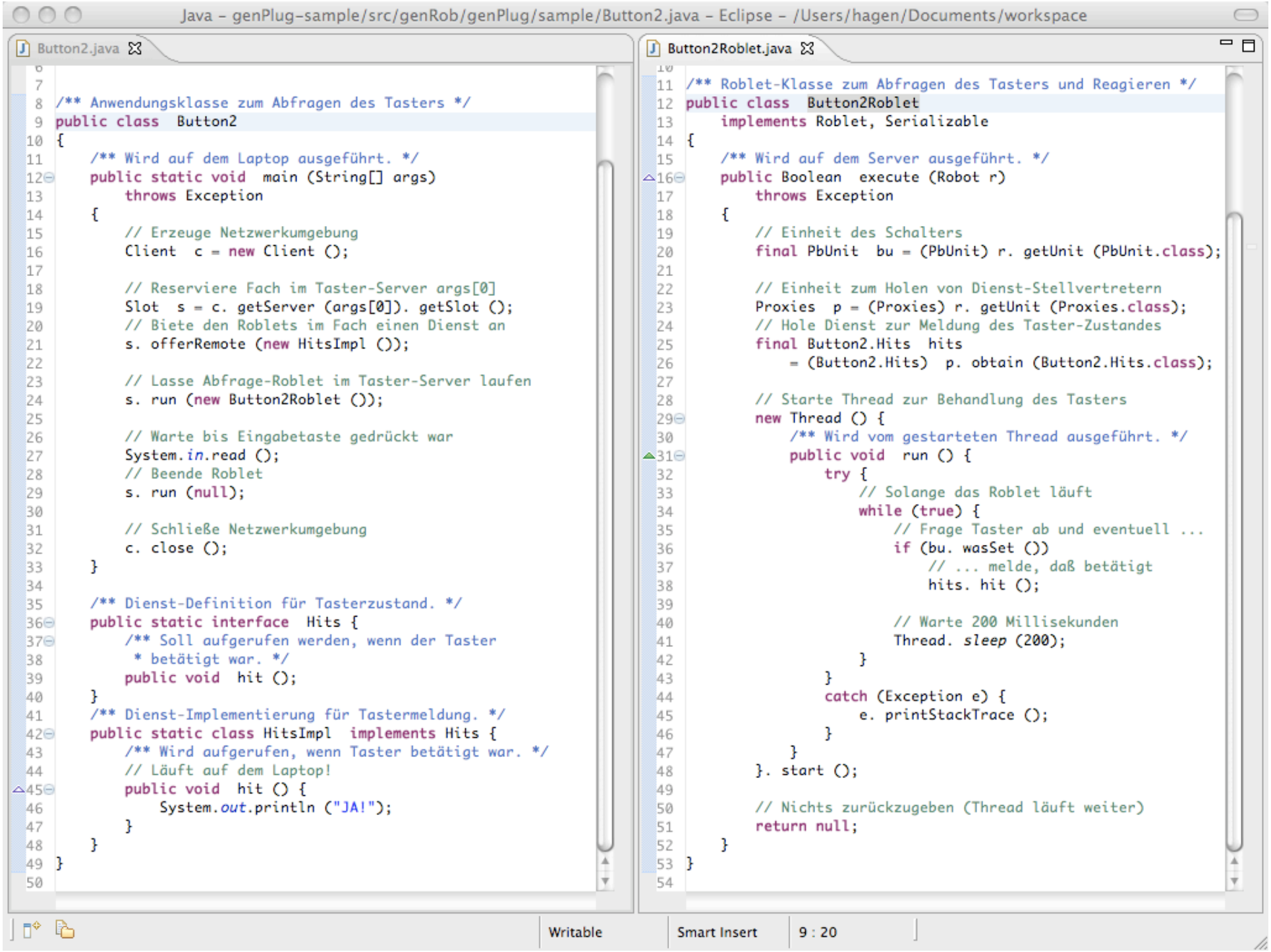

```java
package genRob.genPlug.sample;

import  genRob.genControl.client.Client;
import  genRob.genControl.client.Slot;

/** Anwendungsklasse zum Abfragen des Tasters
 * und gegebenenfalls Umschalten der Rundumleuchte */
public class  Central
{
    /** Wird auf dem Laptop ausgeführt. */
    public static void  main (String[] args)
        throws Exception
    {
        // Erzeuge Netzwerkumgebung
        Client  c = new Client ();

        // Reserviere Fächer in Servern args[0,1]
        Slot  bs = c. getServer (args[0]). getSlot ();
        Slot  os = c. getServer (args[1]). getSlot ();

        // Erzeuge Roblet-Instanzen für Abfrage und Umschalten
        ButtonRoblet  br = new ButtonRoblet ();
        OutletRoblet  or = new OutletRoblet ();

        // Wiederhole solange nicht Eingabetaste gedrückt war
        while (System.in. available () == 0)
        {
            // Lasse Abfrage-Roblet im Taster-Server laufen
            // und prüfe gleich Rückgabewert
            if (((Boolean) bs. run (br)). booleanValue ())
                // Der Schalter war betätigt worden und
                // nun lasse Umschalt-Roblet im
                // Leuchten-Server laufen
                os. run (or);

            // Warte 200 Millisekunden
            Thread. sleep (200);
        }

        // Schließe Netzwerkumgebung
        c. close ();
    }
}
```

```java
/** Roblet-Klasse zum Abfragen des Schalters */
public class  ButtonRoblet
    implements Roblet, Serializable
{
    /** Wird auf dem Server ausgeführt. */
    public Boolean  execute (Robot r)
        throws Exception
    {
        // Einheit des Schalters
        PbUnit  bu = (PbUnit) r. getUnit (PbUnit.class);

        // Hole, ob betätigt war, und gibt als Resultat
        // zurück
        return new Boolean (bu. wasSet ());
    }
}
```

```java
/** Roblet-Klasse zum Umschalten der Rundumleuchte */
class  OutletRoblet
    implements Roblet, Serializable
{
    /** Wird auf dem Server ausgeführt. */
    public Object  execute (Robot r)
        throws Exception
    {
        // Einheit der Steckdosenleiste
        SispmUnit  su
                = (SispmUnit) r. getUnit (SispmUnit.class);

        // Adresse der Steckdosenleiste mit der Leuchte
        SispmSerial  ss
            = new SispmSerial (0x01, 0x01, 0x4b, 0xc3, 0xf2);
        // Steckplatz der Leuchte in der Steckdosenleiste
        int  outlet = 0;

        // Gesetzten Zustand erfragen (ein oder aus)
        boolean  isOn = su. isOn (ss, outlet, true);
        // Umschalten (ein->aus oder aus-> ein)
        su. setOn (ss, outlet, ! isOn);

        // Nichts zurückzugeben
        return null;
    }
```

Teil 2 – Notschalter abfragen 2 (`Button2.java`)

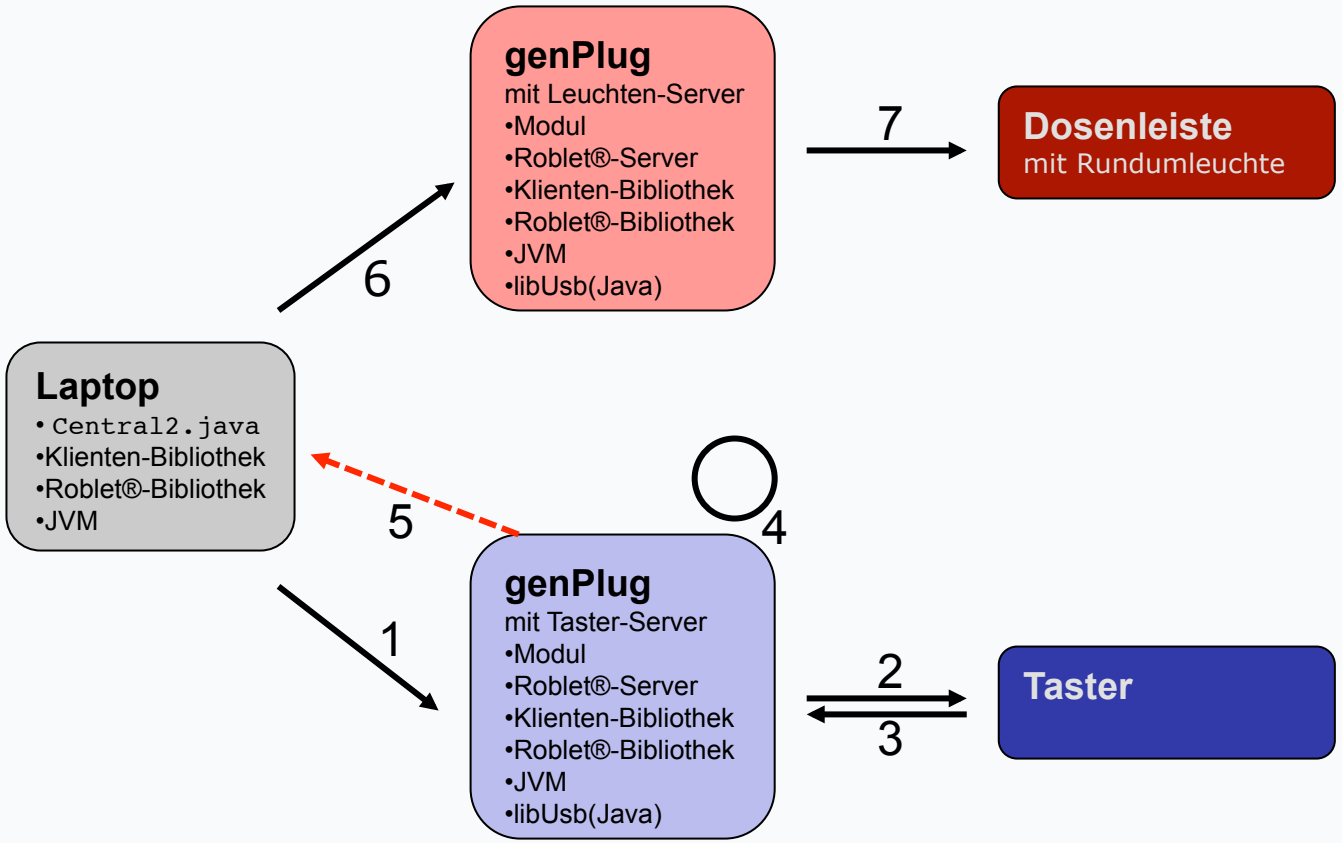




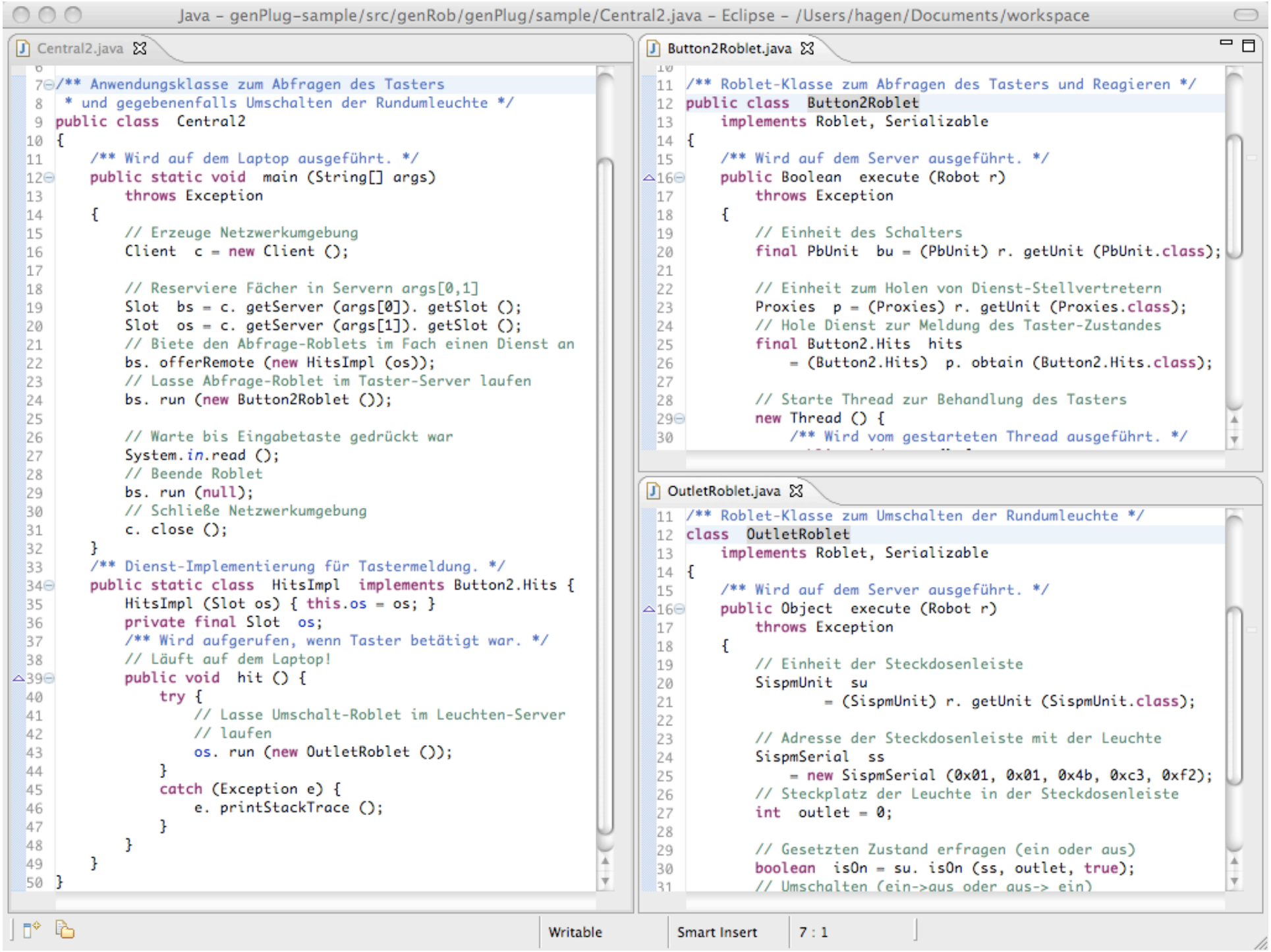

```java
/** Anwendungsklasse zum Abfragen des Tasters */
public class  Button2
{
    /** Wird auf dem Laptop ausgeführt. */
    public static void  main (String[] args)
        throws Exception
    {
        // Erzeuge Netzwerkumgebung
        Client  c = new Client ();

        // Reserviere Fach im Taster-Server args[0]
        Slot  s = c. getServer (args[0]). getSlot ();
        // Biete den Roblets im Fach einen Dienst an
        s. offerRemote (new HitsImpl ());

        // Lasse Abfrage-Roblet im Taster-Server laufen
        s. run (new Button2Roblet ());

        // Warte bis Eingabetaste gedrückt war
        System.in.read ();
        // Beende Roblet
        s. run (null);

        // Schließe Netzwerkumgebung
        c. close ();
    }

    /** Dienst-Definition für Tasterzustand. */
    public static interface  Hits {
        /** Soll aufgerufen werden, wenn der Taster
         * betätigt war. */
        public void  hit ();
    }
    /** Dienst-Implementierung für Tastermeldung. */
    public static class HitsImpl  implements Hits {
        /** Wird aufgerufen, wenn Taster betätigt war. */
        // Läuft auf dem Laptop!
        public void  hit () {
            System.out.println ("JA!");
        }
    }
}
```

```java
/** Roblet-Klasse zum Abfragen des Tasters und Reagieren */
public class  Button2Roblet
    implements Roblet, Serializable
{
    /** Wird auf dem Server ausgeführt. */
    public Boolean  execute (Robot r)
        throws Exception
    {
        // Einheit des Schalters
        final PbUnit  bu = (PbUnit) r. getUnit (PbUnit.class);

        // Einheit zum Holen von Dienst-Stellvertretern
        Proxies  p = (Proxies) r. getUnit (Proxies.class);
        // Hole Dienst zur Meldung des Taster-Zustandes
        final Button2.Hits  hits
            = (Button2.Hits)  p. obtain (Button2.Hits.class);

        // Starte Thread zur Behandlung des Tasters
        new Thread () {
            /** Wird vom gestarteten Thread ausgeführt. */
            public void  run () {
                try {
                    // Solange das Roblet läuft
                    while (true) {
                        // Frage Taster ab und eventuell ...
                        if (bu. wasSet ())
                            // ... melde, daß betätigt
                            hits. hit ();

                        // Warte 200 Millisekunden
                        Thread. sleep (200);
                    }
                }
                catch (Exception e) {
                    e. printStackTrace ();
                }
            }
        }. start ();

        // Nichts zurückzugeben (Thread läuft weiter)
        return null;
    }
}
```

Teil 2 – Zentrale Steuerung 2 (`Central2.java`)



http://genRob.com/genPlug

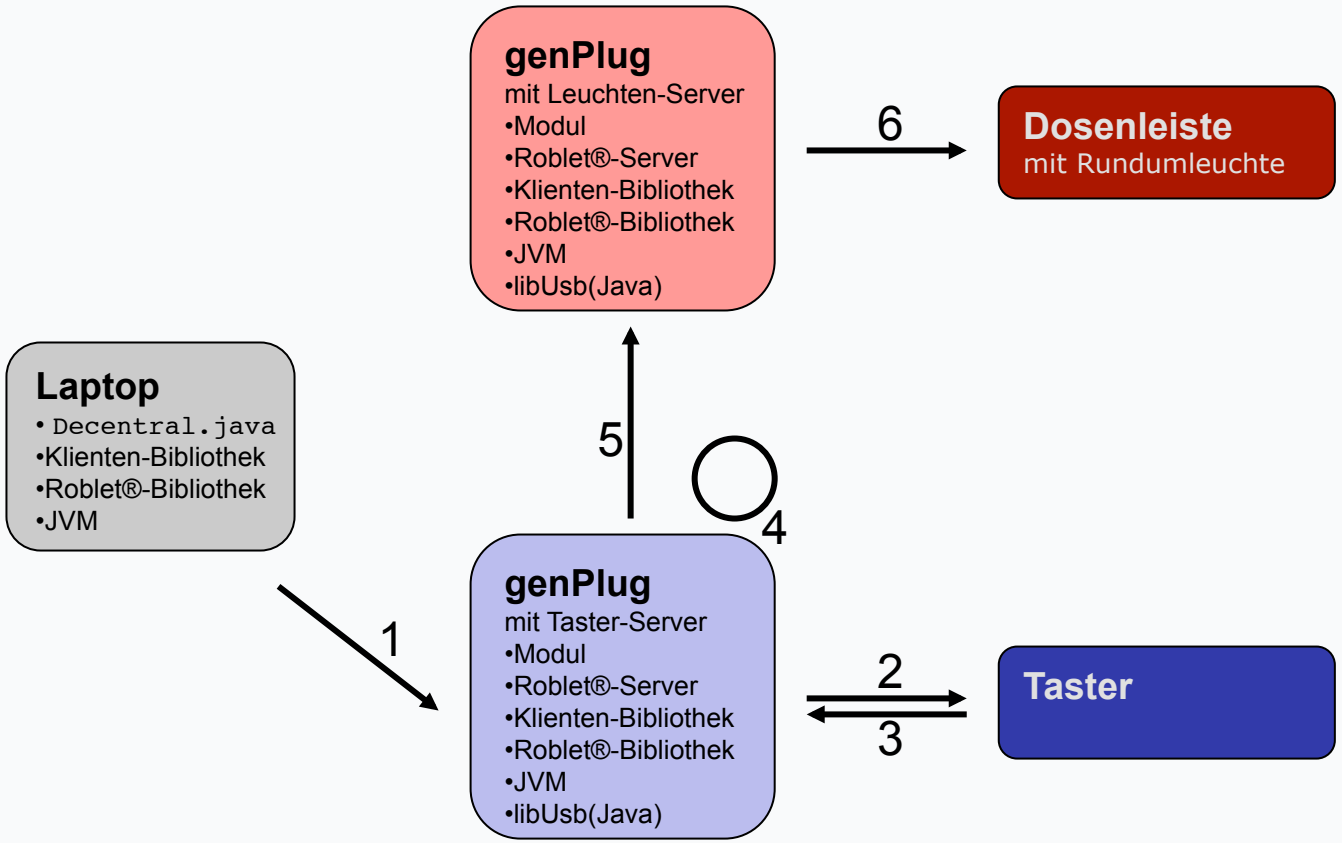



```java
/** Anwendungsklasse zum Abfragen des Tasters
 * und gegebenenfalls Umschalten der Rundumleuchte */
public class  Central2
{
    /** Wird auf dem Laptop ausgeführt. */
    public static void  main (String[] args)
        throws Exception
    {
        // Erzeuge Netzwerkumgebung
        Client  c = new Client ();

        // Reserviere Fächer in Servern args[0,1]
        Slot  bs = c. getServer (args[0]). getSlot ();
        Slot  os = c. getServer (args[1]). getSlot ();
        // Biete den Abfrage-Roblets im Fach einen Dienst an
        bs. offerRemote (new HitsImpl (os));
        // Lasse Abfrage-Roblet im Taster-Server laufen
        bs. run (new Button2Roblet ());

        // Warte bis Eingabetaste gedrückt war
        System.in.read ();
        // Beende Roblet
        bs. run (null);
        // Schließe Netzwerkumgebung
        c. close ();
    }
    /** Dienst-Implementierung für Tastermeldung. */
    public static class  HitsImpl  implements Button2.Hits {
        HitsImpl (Slot os) { this.os = os; }
        private final Slot  os;
        /** Wird aufgerufen, wenn Taster betätigt war. */
        // Läuft auf dem Laptop!
        public void  hit () {
            try {
                // Lasse Umschalt-Roblet im Leuchten-Server
                // laufen
                os. run (new OutletRoblet ());
            }
            catch (Exception e) {
                e. printStackTrace ();
            }
        }
    }
}
```

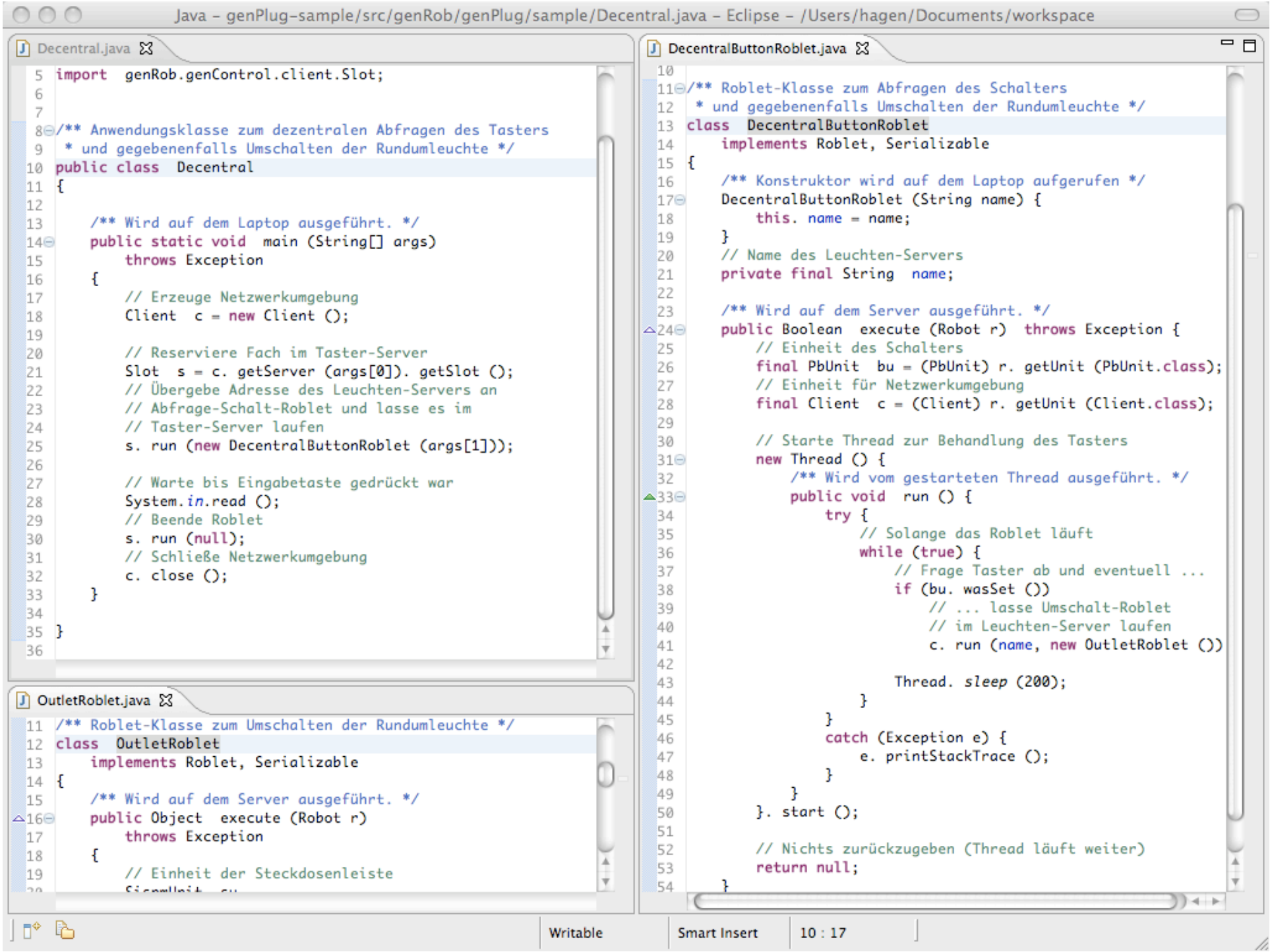
```java
/** Roblet-Klasse zum Abfragen des Tasters und Reagieren */
public class  Button2Roblet
    implements Roblet, Serializable
{
    /** Wird auf dem Server ausgeführt. */
    public Boolean  execute (Robot r)
        throws Exception
    {
        // Einheit des Schalters
        final PbUnit  bu = (PbUnit) r. getUnit (PbUnit.class);

        // Einheit zum Holen von Dienst-Stellvertretern
        Proxies  p = (Proxies) r. getUnit (Proxies.class);
        // Hole Dienst zur Meldung des Taster-Zustandes
        final Button2.Hits  hits
            = (Button2.Hits)  p. obtain (Button2.Hits.class);

        // Starte Thread zur Behandlung des Tasters
        new Thread () {
            /** Wird vom gestarteten Thread ausgeführt. */
```

```java
/** Roblet-Klasse zum Umschalten der Rundumleuchte */
class  OutletRoblet
    implements Roblet, Serializable
{
    /** Wird auf dem Server ausgeführt. */
    public Object  execute (Robot r)
        throws Exception
    {
        // Einheit der Steckdosenleiste
        SispmUnit  su
                = (SispmUnit) r. getUnit (SispmUnit.class);

        // Adresse der Steckdosenleiste mit der Leuchte
        SispmSerial  ss
            = new SispmSerial (0x01, 0x01, 0x4b, 0xc3, 0xf2);
        // Steckplatz der Leuchte in der Steckdosenleiste
        int  outlet = 0;

        // Gesetzten Zustand erfragen (ein oder aus)
        boolean  isOn = su. isOn (ss, outlet, true);
        // Umschalten (ein->aus oder aus-> ein)
```

Teil 2 – Dezentrale Steuerung (`Decentral.java`)



http://genRob.com/genPlug


Java – genPlug-sample/src/genRob/genPlug/sample/Decentral.java – Eclipse – /Users/hagen/Documents/workspace
Decentral.java
import genRob.genControl.client.Slot;
/** Anwendungsklasse zum dezentralen Abfragen des Tasters
* und gegebenenfalls Umschalten der Rundumleuchte */
public class Decentral
{
/** Wird auf dem Laptop ausgeführt. */
public static void main (String[] args)
throws Exception
{
// Erzeuge Netzwerkumgebung
Client c = new Client ();
// Reserviere Fach im Taster-Server
Slot s = c. getServer (args[0]). getSlot ();
// Übergebe Adresse des Leuchten-Servers an
// Abfrage-Schalt-Roblet und lasse es im
// Taster-Server laufen
s. run (new DecentralButtonRoblet (args[1]));
// Warte bis Eingabetaste gedrückt war
System.in.read ();
// Beende Roblet
s. run (null);
// Schließe Netzwerkumgebung
c. close ();
}
}
OutletRoblet.java
/** Roblet-Klasse zum Umschalten der Rundumleuchte */
class OutletRoblet
implements Roblet, Serializable
{
/** Wird auf dem Server ausgeführt. */
public Object execute (Robot r)
throws Exception
{
// Einheit der Steckdosenleiste
DecentralButtonRoblet.java
/** Roblet-Klasse zum Abfragen des Schalters
* und gegebenenfalls Umschalten der Rundumleuchte */
class DecentralButtonRoblet
implements Roblet, Serializable
{
/** Konstruktor wird auf dem Laptop aufgerufen */
DecentralButtonRoblet (String name) {
this. name = name;
}
// Name des Leuchten-Servers
private final String name;
/** Wird auf dem Server ausgeführt. */
public Boolean execute (Robot r) throws Exception {
// Einheit des Schalters
final PbUnit bu = (PbUnit) r. getUnit (PbUnit.class);
// Einheit für Netzwerkumgebung
final Client c = (Client) r. getUnit (Client.class);
// Starte Thread zur Behandlung des Tasters
new Thread () {
/** Wird vom gestarteten Thread ausgeführt. */
public void run () {
try {
// Solange das Roblet läuft
while (true) {
// Frage Taster ab und eventuell ...
if (bu. wasSet ())
// ... lasse Umschalt-Roblet
// im Leuchten-Server laufen
c. run (name, new OutletRoblet ())
Thread. sleep (200);
}
}
catch (Exception e) {
e. printStackTrace ();
}
}
}. start ();
// Nichts zurückzugeben (Thread läuft weiter)
return null;
}
Writable
Smart Insert
10 : 17

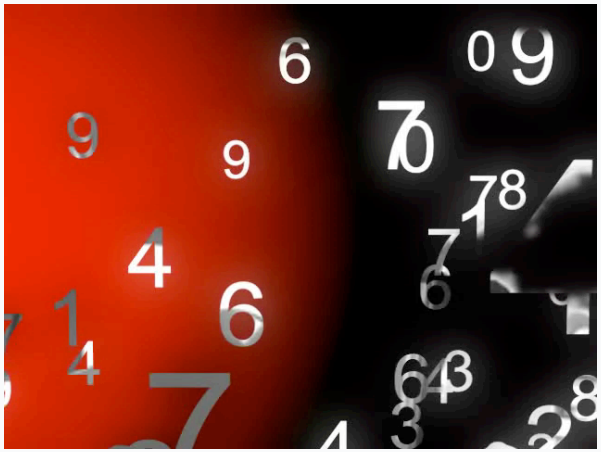
Teil 3

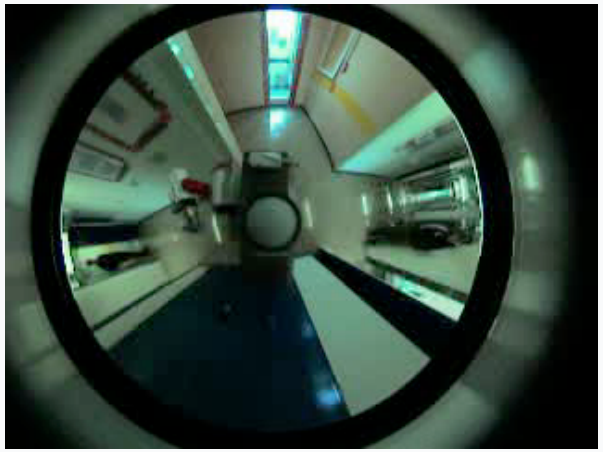
Anwendungen, Zusammenfassung

Teil 3 – Anwendungen 1

http://www.uni-bielefeld.de/biologie/biochem/index.html

Teil 3 – Anwendungen 2





http://tams-www.informatik.uni-hamburg.de/research/robotics/service_robot/videos/index.php

Teil 3 – Anwendungen 3

http://tams-www.informatik.uni-hamburg.de/research/robotics/service_robot/videos/index.php

Teil 3 – Anwendungen 4

genRob®-System

- genMediator (Verzeichnisdienst)
- genMap (Kartendienst)
- genPath (Bahnplanungsdienst)
- 2x genSimulation (Robotersimulationen)
- genView (Grafische Bedienschnittstelle)

Teil 3 – Anwendungen 5

Z-Windows

Teil 3 – Anwendungen 6

Module können beliebige Ressourcen darstellen

- Dateien und Dateisysteme
- Datenbank-Zugriffe
- jegliche Art von Netzarbeit (Roblets®, RMI, TCP, SOAP, ...)
- Im Grunde jede Ressource, die über (andere) Bibliotheken zugreifbar ist – insbesondere über JNI

Im Grunde kann jegliche Ressource zur Verfügung gestellt werden!

Teil 3 – Anwendungen 7

cloud computing

Teil 3 – Zusammenfassung 1

- Mit der Roblet®-Technik verschwindet der Bedarf an komplexer Funktionalität im Server.
- Wenige Zeilen ohne Konfigurationsdateien, schon arbeitet man verteilt und kann leicht entscheiden, wo der Code läuft.

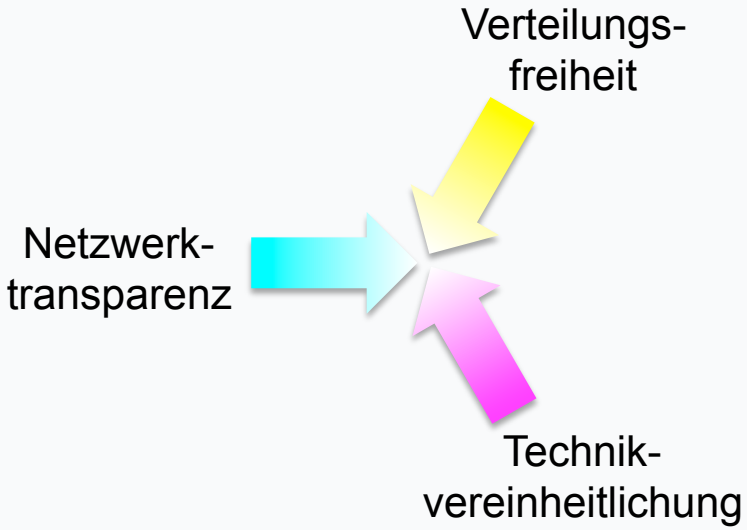
Teil 3 – Zusammenfassung 2

Alles geht!

- beliebige Architekturen und Paradigmen wie SOA u.a.
- beliebige Versionierungssysteme wie SVN, ClearCase
- beliebige Editoren und Entwicklungsumgebungen wie Eclipse, NetBeans
- beliebige Bausysteme wie Ant, Maven, Gradle
- Zusammenspiel mit beliebige anderen Techniken inklusive RMI, OSGi, Servlet-Container, J2EE etc.

„Ich habe ein Produkt und möchte nur einen Teil auf Roblet®-Technik umstellen!“ –> GEHT!

Teil 3 – Zusammenfassung – Effizienzdreieck



Vielen Dank

WHOI / Sentry – Foto: University of Washington

Kontakt:

- http://roblet.org

- Hagen Stanek
- stanek@roblet.org
- ++49 7034 251692

- http://genRob.com

Ausprobieren der Roblet®-Technik

- roblet.org-WLAN
- http://10.10.10.10
- Folgen Sie den Hinweisen
- Fragen Sie mich ...

Kontakt:

- http://roblet.org

- Hagen Stanek
- stanek@roblet.org
- ++49 7034 251692

- http://genRob.com